eBook Gratuit

APPRENEZ gitlab

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#gitlab

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: gitlab

It is an unofficial and free gitlab ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official gitlab.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec gitlab

Remarques

GitLab est un logiciel de contrôle de version basé sur le Web, basé sur git, qui ajoute des fonctionnalités supplémentaires telles que le suivi des bogues de gestion des succursales et l'intégration continue. Il est développé en Ruby.

GitLab Community Edition (CE) est en cours de développement open-source et utilise la licence MIT. Depuis août 2013, GitLab Inc. fournit l' *édition Enterprise Enterprise Edition (EE)* qui inclut plus de fonctionnalités que l'édition communautaire.

GitLab.com est l'alternative Software as a Service et s'exécute sur une édition Enterprise. Il est gratuit et prend en charge les référentiels privés et publics.

Versions

Version	Date de sortie
8.15	2016-12-22
8.14	2016-11-22
8.13	2016-10-22
8.12	2016-09-22
8.11	2016-08-22
8.10	2016-07-22
8,9	2016-06-22
8,8	2016-05-22
8.7	2016-04-22
8.6	2016-03-22
8.5	2016-02-22
8.4	2016-01-22
8.3	2015-12-22
8.2	2015-11-22
8.1	2015-10-22

Version	Date de sortie
8.0	2015-09-22

Examples

Installation ou configuration

Ceci est un bref résumé du guide GitLab sur l' installation d'un package GitLab CE Omnibus .

Exigences

Pour installer GitLab Community Edition sur votre serveur, vous devez lire la page relative aux exigences . Pour être bref, les exigences recommandées sont les suivantes:

- **OS:** Ubuntu, Debian, CentOS, RHEL
- **Version Ruby:** Ruby (MRI) 2.1.x, ne fonctionne pas actuellement avec les versions 2.2 ou 2.3.
- **CPU:** 2 cœurs (prend en charge jusqu'à 500 utilisateurs)
- **Mémoire:** 2 Go (prend en charge jusqu'à 100 utilisateurs)
- **Base de données:** PostgreSQL

Installation

La méthode recommandée consiste à installer le package Omnibus, rapide à installer. Il contient GitLab et toutes ses dépendances (Ruby, PostgreSQL, Redis, Nginx, Unicorn, etc.). Pour d'autres méthodes, consultez les options d'installation de GitLab.

Avec Ubuntu 16.04 comme système d'exploitation recommandé, ce guide décrit les étapes d'installation des distributions basées sur Debian. Pour CentOS, RHEL, Oracle Linux et Scientific Linux, veuillez vous reporter aux guides originaux:

- CentOS 6 (et RedHat / Oracle / Scientific Linux 6)
- CentOS 7 (et RedHat / Oracle / Scientific Linux 7)

Ubuntu, Debian, Raspberrian

Installez les dépendances nécessaires. Si vous utilisez Postfix, sélectionnez "Site Internet" lors de la configuration

```
sudo apt-get install curl openssh-server ca-certificates postfix apt-transport-https
curl https://packages.gitlab.com/gpg.key | sudo apt-key add -
```

Ajoutez le serveur de packages Gitlab et installez le package

```
sudo curl -sS https://packages.gitlab.com/install/repositories/gitlab/raspberry-
pi2/script.deb.sh | sudo bash
sudo apt-get install gitlab-ce
```

Si vous ne souhaitez pas installer le référentiel via un script, téléchargez le package manuellement et installez-le en utilisant

```
dpkg -i gitlab-ce_<version>.deb
```

Maintenant, configurez et démarrez GitLab

```
sudo gitlab-ctl reconfigure
```

Enfin, naviguez jusqu'au nom d'hôte et connectez-vous. Au début, vous serez redirigé pour fournir un mot de passe pour le compte administrateur initial. Après cela, vous pouvez vous connecter. Le **nom d'utilisateur du compte d'administrateur par défaut** est **root** .

Lire Démarrer avec gitlab en ligne: https://riptutorial.com/fr/gitlab/topic/2046/demarrer-avec-gitlab

Chapitre 2: Configuration CI Android

Examples

Build Tools 24.0.0 - Android

```
image: jangrewe/gitlab-ci-android
before_script:
  - apt-get --quiet update --yes
  - apt-get --quiet install --yes wget tar unzip lib32stdc++6 lib32z1 openjdk-8-jdk
  - echo y | ${ANDROID_HOME}/tools/android --silent update sdk --no-ui --all --filter android-
24
  - echo y | ${ANDROID_HOME}/tools/android --silent update sdk --no-ui --all --filter
platform-tools
  - echo y | ${ANDROID_HOME}/tools/android --silent update sdk --no-ui --all --filter build-
tools-24.0.0
  - echo y | ${ANDROID_HOME}/tools/android --silent update sdk --no-ui --all --filter extra-
android-m2repository
  - echo y | ${ANDROID_HOME}/tools/android --silent update sdk --no-ui --all --filter extra-
google-google_play_services
  - echo y | ${ANDROID_HOME}/tools/android --silent update sdk --no-ui --all --filter extra-
google-m2repository
  - chmod +x gradlew
build:
  script:
    - ./gradlew assembleDebug
  artifacts:
    paths:
    - app/build/outputs/apk/app-debug.apk
```

Remplacez le numéro des outils de génération par votre cible de compilation et utilisez l'image du docker si vous ne voulez pas tout installer à chaque fois.

Lire Configuration CI Android en ligne: https://riptutorial.com/fr/gitlab/topic/6952/configuration-ci-android

Chapitre 3: Configuration de Google Cloud SDK CI

Examples

Exemple de Docker JDK 8 ouvert

```
  image: openjdk:8-jdk

  before_script:
       - curl https://dl.google.com/dl/cloudsdk/release/google-cloud-sdk.tar.gz > /tmp/google-
cloud-sdk.tar.gz
       - mkdir -p /usr/local/gcloud
       - tar -C /usr/local/gcloud -xvf /tmp/google-cloud-sdk.tar.gz
       - echo y |/usr/local/gcloud/google-cloud-sdk/install.sh
       - chmod +x ./gradlew

    build:
      script:
        - ./gradlew build
      artifacts:
        paths:
        - app/build/outputs/
```

Lire Configuration de Google Cloud SDK CI en ligne:
https://riptutorial.com/fr/gitlab/topic/9094/configuration-de-google-cloud-sdk-ci

Chapitre 4: Intégration continue

Introduction

Le CI GitLab exécute les travaux de construction basés sur un `.gitlab-ci.yml` . Les travaux sont exécutés sur un serveur distant dans son propre conteneur Docker.

Le serveur CI lui-même est configuré avec un `config.toml` .

Remarques

- Une construction échouera si des lignes d'un travail renvoient un code de sortie! = 0.

Examples

Installation de coureur

Debian, Ubuntu et CentOS

1. Ajouter le dépôt officiel

Debian / Ubuntu

```
curl -L https://packages.gitlab.com/install/repositories/runner/gitlab-ci-multi-runner/script.deb.sh | sudo bash
```

CentOS

```
curl -L https://packages.gitlab.com/install/repositories/runner/gitlab-ci-multi-runner/script.rpm.sh | sudo bash
```

2. Installez le `gitlab-ci-multi-runner`

Debian / Ubuntu

```
sudo apt-get install gitlab-ci-multi-runner
```

CentOS

```
sudo yum install gitlab-ci-multi-runner
```

3. Enregistrez le coureur

```
sudo gitlab-ci-multi-runner register
```

- Entrez l'URL de votre CI GitLab. Cela devrait ressembler à ceci: `http://example.com/ci`
- Entrez le jeton d'enregistrement. S'il s'agit d'un runner spécifique au projet, vous pouvez trouver le jeton dans `Project settings -> Runners` . S'il s'agit d'un coureur partagé, accédez à la `Admin area -> Runners` et trouvez le jeton d'inscription à cet `Admin area -> Runners` .
- Donnez maintenant à votre coureur un nom descriptif.
- Sélectionnez l'exécuteur que vous souhaitez utiliser. Les exécuteurs valides sont: `shell` (ceux-ci peuvent être configurés ultérieurement pour utiliser sh ou bash), `docker` , `docker-ssh` , `ssh` , `parallels` , `virtualbox` , `docker+machine` ou `docker-ssh+machine` . Pour plus d'informations sur les exécuteurs, consultez la documentation officielle .

les fenêtres

1. Téléchargez le binaire du coureur et placez-le dans votre système.
2. Ouvrez une invite de commande en tant qu'administrateur
3. Enregistrez le coureur

```
<runner-binary> register
```

- Entrez l'URL de votre CI GitLab. Cela devrait ressembler à ceci: `http://example.com/ci`
- Entrez le jeton d'enregistrement. S'il s'agit d'un runner spécifique au projet, vous pouvez trouver le jeton dans `Project settings -> Runners` . S'il s'agit d'un coureur partagé, accédez à la `Admin area -> Runners` et trouvez le jeton d'inscription à cet `Admin area -> Runners` .
- Donnez maintenant à votre coureur un nom descriptif.
- Sélectionnez l'exécuteur que vous souhaitez utiliser. Les exécuteurs valides sont: `shell` (peut être configuré ultérieurement pour utiliser cmd ou powershell), `ssh` , `parallels` ou `virtualbox` . Pour plus d'informations sur les exécuteurs, consultez la documentation officielle .

4. (Facultatif) Enregistrez le coureur en tant que service

```
<runner-binary> install --user <username> --password <password>
```

5. Commencer le coureur

```
<runner-binary> start
```

Configuration du coureur

L'emplacement de configuration de votre coureur est le suivant:

Debian / Ubuntu / CentOS

`/etc/gitlab-runner/config.toml` si exécuté en tant que root

`~/.gitlab-runner/config.toml` si exécuté en tant que non root

les fenêtres

`config.toml` où se trouve votre binaire

Un `config.toml` minimal peut ressembler à ceci:

```
concurrent = 1
[[runners]]
  name = "ExampleRunner"
  url = "https://example.com/ci"
  token = "f3058595ca4b2d217726466b1feed9"
  executor = "shell"
  shell = "bash"
```

Pour une configuration avancée, veuillez consulter la documentation officielle .

Configuration de Gitlab CI pour permettre le clonage d'autres référentiels privés

Certains projets comme GoLang peuvent avoir besoin de cloner d'autres référentiels GitLab dépendants pendant la génération. Pour que cela fonctionne, vous pouvez ajouter une clé de déploiement aux référentiels dépendants et placer la clé privée (sans mot de passe) dans le référentiel d'origine.

Créez et archivez une clé SSH dans le référentiel Git qui dépend d'un autre référentiel au moment de la construction:

```
ssh-keygen -t rsa -b 4096 -C "My CI Deploykey"

# In the following promt name the key "deploykey" and leave the passphrase empty
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa): deploykey
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in deploykey.
Your public key has been saved in deploykey.pub.

# check-in both files
```

Utilisez `deploykey.pub` pour configurer une clé de déploiement dans le référentiel dépendant. Vous pouvez trouver une page Deploykey dans les paramètres du projet GitLab.

Maintenant, ajoutez ce qui suit à vous `.gitlab-ci.yml`

```
before_script:
  # Git and SSH setup to clone private repos
```

```
# Needs the deploykey file to be installed in all dependent repositories
- git config --global url."git@gitlab.com:".insteadOf "https://gitlab.com/"
# Add gitlab to known_hosts
- mkdir -p ~/.ssh && chmod 700 ~/.ssh
- ssh-keyscan -H gitlab.com >> ~/.ssh/known_hosts
# Start the ssh agent and add the deploykey
- chmod 400 deploykey
- eval $(ssh-agent -s)
- ssh-add deploykey
```

Maintenant, tout appel à `git clone` dans votre build devrait fonctionner. Même si c'est via d'autres outils comme `go get` , `govendor sync` ou tout ce que vous utilisez.

Lire Intégration continue en ligne: https://riptutorial.com/fr/gitlab/topic/6258/integration-continue

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec gitlab	Alessandro Trinca Tornidor, Community, Connor Shea, Fairy, Greg Dubicki, jim
2	Configuration CI Android	ReverseCold
3	Configuration de Google Cloud SDK CI	Michael Meyer
4	Intégration continue	Fairy, Tarion